Chapter 35: Functions used in Actions

There is a set of supported functions that can be used in action stanzas. Actions are described in Chapter 34: *Actions and ACTION Keyword Values*. In the present chapter we give a general overview of functions, list and describe all the supported functions, provide a couple of examples of functions within actions, and list all the read-only variables available to the supported functions.

35.1 Overview of Functions

Table files and **UPD** configuration files often include actions. An action corresponds to a command, usually a **UPS** command, and lists functions to perform in addition to the command's internal processes, when the command is executed. The supported functions are listed and described in this chapter. A function has the format:

```
<function_name>([<argument_1>] [, <argument_2>] ... [<delimiter>])
```

The default delimiter is the colon (`:`).

For example, the function:

```
envPrepend(<VARIABLE>, <value>)
```

prepends the specified value to an existing environment variable, using the default delimiter.

Functions are not case-sensitive; e.g., **`envPrepend`**, **`envprepend`**, and **`ENVPREPEND`** are all acceptable and equivalent. A function is specified in a shell-independent manner, but contains enough information to allow it to be transformed into a **sh** or **csh** family command (e.g., **`sourceRequired()`**, or **`execute()`**), or to be interpreted directly by **UPS** (e.g., **`writeCompileScript()`**).

35.2 Reversible Functions

In section 34.2.2 *"Uncommands" as Actions* we discussed commands that have corresponding "uncommands". Usually, when the "uncommand" is run, the desired behavior is to reverse all the functions that were performed when the original command was run. Many of the supported functions are reversible, some are not.

Wherever you plan to default the "uncommand" action (i.e., to specifically *not* include an ACTION=UNCOMMAND stanza) and you want **UPS** to exactly reverse the ACTION=COMMAND functions, make sure that you only include reversible functions under ACTION=COMMAND. Reversible functions are identified as such in the descriptions in section 35.3 *Function Descriptions*.

35.3 Function Descriptions

35.3.1 addAlias

Description

Add an alias (C shell family) or function (Bourne shell family). A **`%s`** in the **`<VALUE>`** marks where the argument list should go. Reversible (runs **`unAlias`**).

Syntax

```
addAlias(<NAME>, <VALUE>)
```

Example 1

```
addAlias(askfor, 'echo May I have some %s, please\?')
```

Defines the alias **`askfor`**, which when run with an argument like **`cake`**, e.g.,:

```
% askfor cake
```

produces the response:

```
May I have some cake, please?
```

Example 2

```
addAlias(setup,'${UPS_SOURCE} `${UPS_PROD_DIR}/bin/ups setup %s`')
```

${UPS_SOURCE} is set to "**.**" or "**source**" depending on the shell, and **%s** is presumed to stand for a product name. This defines the alias **setup**. When issued with a product name, e.g.,

```
% setup upd
```

it sources the executable `${UPS_PROD_DIR}/bin/ups` with the arguments **setup** and **upd**.

35.3.2 doDefaults

Description

Perform the default functions for the command corresponding to the specified action (only SETUP and UNSETUP have default functions). If no action listed (e.g., **doDefaults()**), then the action under which this function occurs is used. Reversible (runs **doDefaults**).

Note: If an ACTION corresponding to the given command is included in the file, the command's default functions will be executed only if **doDefaults** is specified underneath it. If there is no ACTION for the command, and hence no **doDefaults** function listed, the default functions will be executed when the command is issued.

Syntax

```
doDefaults([<ACTION>])
```

Example

```
doDefaults([SETUP])
```

Specifies that the default functions for the **setup** command will be run when the command is issued. More typically, this is specified in the following manner:

```
ACTION=SETUP
       doDefaults()
```

35.3.3 Else

Description

A conditional, to be used with **If** and **EndIf** or with **Unless** and **EndUnless**. **Else** takes no command string. **Else** is optional, but we recommend including it for clarity it even if no code follows it. See sections 35.3.17 *If* and 35.3.33 *Unless* for descriptions of processing. Also see sections 35.3.4 *EndIf* and 35.3.5 *EndUnless*.

Syntax

Else()

Example

```
Action=Setup
ProdDir()
SetupEnv()
EnvSetIfNotSet(FOO,":")
EnvPrepend(FOO, ${UPS_PROD_DIR}/basic)

If( test -d ${UPS_PROD_DIR}/exciting )
  EnvPrepend(FOO, ${UPS_PROD_DIR}/exciting)
Else()
  EnvPrepend(FOO, ${UPS_PROD_DIR}/boring)
EndIf( test -d ${UPS_PROD_DIR}/exciting )
```

35.3.4 EndIf

Description

Closes a conditional; to be used with **If** and optionally **Else**. The **EndIf** statement must include a command that exactly matches the command in the corresponding **If** statement. This is because **UPS** must be able to invert this to get an unsetup action (to unsetup, the order gets inversed and the inverse of the actions are called). Also see sections 35.3.17 *If* and 35.3.3 *Else*.

Syntax

EndIf(<same command as used in If statement>)

Example

```
Action=Setup
```

```
ProdDir()
SetupEnv()
EnvSetIfNotSet(FOO,":")
EnvPrepend(FOO, ${UPS_PROD_DIR}/basic)

If( test -d ${UPS_PROD_DIR}/exciting )
  EnvPrepend(FOO, ${UPS_PROD_DIR}/exciting)
Else()
  EnvPrepend(FOO, ${UPS_PROD_DIR}/boring)
EndIf( test -d ${UPS_PROD_DIR}/exciting )
```

35.3.5 EndUnless

Description

Closes a conditional; to be used with **`Unless`** and optionally **`Else`**. The **`EndUnless`** statement must include a command that exactly matches the command in the corresponding **`If`** statement. This is because **UPS** must be able to invert this to get an unsetup action (to unsetup, the order gets inversed and the inverse of the actions are called). Also see sections 35.3.33 *Unless* and 35.3.3 *Elseif*.

Syntax

`EndUnless(<same command as used in Unless statement>)`

Example

```
Action=Setup
ProdDir()
SetupEnv()
EnvSetIfNotSet(FOO,":")
EnvPrepend(FOO, ${UPS_PROD_DIR}/basic)

Unless( test -d ${UPS_PROD_DIR}/exciting )
  EnvPrepend(FOO, ${UPS_PROD_DIR}/boring)
Else()
  EnvPrepend(FOO, ${UPS_PROD_DIR}/exciting)
EndUnless( test -d ${UPS_PROD_DIR}/exciting )
```

35.3.6 envAppend

Description

Append **<value>** to existing environment variable. Reversible (runs **envRemove**).

It is better to append than prepend if you just want to provide a value in case one is not there. If you want to override any existing value, you should prepend.

Note: Use the function **pathAppend** for $PATH.

Syntax

```
envAppend(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
envAppend(PYTHONPATH, ${UPS_PROD_DIR}/lib)
```

Appends the value of **${UPS_PROD_DIR}/lib** to the variable **PYTHONPATH**, using the default delimiter.

35.3.7 envPrepend

Description

Prepend **<value>** to existing environment variable. Reversible (runs **envRemove**).

It is better to prepend than append if you want to override any existing value. If you just want to provide a value in case one is not there, you should append.

Note: Use the function **pathPrepend** for $PATH.

Syntax

```
envPrepend(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
envPrepend(PYTHONPATH, ${UPS_PROD_DIR}/lib)
```

Prepends the value of **${UPS_PROD_DIR}/lib** to the variable **PYTHONPATH**, using the default delimiter.

35.3.8 envRemove

Description

Remove the string **<value>** from existing environment variable.

Note: Use the function **pathRemove** for $PATH.

Syntax

```
envRemove(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
envRemove(PYTHONPATH, ${UPS_PROD_DIR}/lib)
```

Removes the value of **${UPS_PROD_DIR}/lib** from the variable **PYTHONPATH**; assumes the default delimiter.

35.3.9 envSet

Description

Set a new environment variable. This is particularly useful for representing long strings. Reversible (runs **envUnset**).

Note: Use the function **pathSet** for $PATH.

Syntax

```
envSet(<VARIABLE>, <value>)
```

Example

```
envSet(UPD_USERCODE_DIR, ${UPS_THIS_DB})
```

Sets ${UPD_USERCODE_DIR} (the local database used by **UPD**) to ${UPS_THIS_DB} (the database in which the product is declared).

35.3.10 envSetIfNotSet

Description

Set a new environment variable, if not already set. This is particularly useful for representing long strings.

Syntax

```
envSetIfNotSet(<VARIABLE>, <value>)
```

Example

```
envSetIfNotSet(HOST, 'long_hostname')
```

If not already set, this sets the variable **HOST** to a long hostname.

35.3.11 envUnset

Description

Unset existing environment variable.

Syntax

```
envUnset(<VARIABLE>)
```

Example

```
envUnset(MYVAR)
```

Unsets the variable $MYVAR.

35.3.12 exeAccess

Description

Check for access to specified existing executable through the $PATH. If executable is found continue. If not found, exit with error.

Syntax

```
exeAccess(<executable>)
```

Example

```
exeAccess(gcc)
```

Ensures that a version of the product **gcc** is in your $PATH.

35.3.13 exeActionOptional

Description

Process the functions associated with the specified action for the same product instance. Do not fail if the action doesn't exist. Reversible.

Syntax

```
exeActionOptional("<newaction>")
```

Example

```
exeActionOptional("CONFIGURE")
```

Process the functions in CONFIGURE action. If no CONFIGURE action, processing continues.

35.3.14 exeActionRequired

Description

Process the functions associated with the specified action for the same product instance. Fail if it doesn't exist. Reversible.

Syntax

```
exeActionRequired("<newaction>")
```

Example

```
exeActionRequired("CONFIGURE")
```

Process the functions in CONFIGURE action. If no CONFIGURE action, processing fails.

35.3.15 execute

Description

Execute a shell-independent command and (optionally) assign the output to an environment variable, **<VARIABLE>**.

The functions **execute**, **sourceRequired**, **sourceReqCheck**, **sourceOptional**, and **sourceOptCheck** each take a required parameter (**UPS_ENV_FLAG**) which indicates whether to define **UPS** local variables. This parameter can take the following values:

UPS_ENV	define all local **UPS** environment variables before sourcing (the script or command relies on these being defined)
NO_UPS_ENV	do not define the local **UPS** environment variables (the script or command doesn't use them)

If the optional third argument, **<VARIABLE>**, is not specified, then the specified command is executed but the output from that command is not saved. This command does not have to be shell-independent.

Syntax

```
execute("<command>", <UPS_ENV_FLAG>, [, <VARIABLE>])
```

Example

```
execute("echo Call final installation script for
  ${UPS_PROD_NAME} ${UPS_PROD_VERSION}", NO_UPS_ENV)
```

(All on one line.) **UPS** echoes the given text and sources the `current` script for the product.

35.3.16 fileTest

Description

Run a shell test on **<file>**, fail if **<test>** is not true (see **man test**).

Syntax

```
fileTest(<file>, <test> [, <errormessage>])
```

Example

```
fileTest(/, -w, "You must be root to run this command.")
```

This tests for write access in the root directory and returns the shown error message if the test fails.

35.3.17 If

Description

A conditional, to be used with **`EndIf`** and optionally with **`Else`**, in the order **`If(<command>)...Else()...Endif(<command>)`**. We recommend that **`Else`** always be included for clarity, even if no code follows it.

- If the command in the **`If`** statement succeeds, then **UPS** runs the code following the **`If`** statement and preceding any **`Else`** and/or **`EndIf`** statement.
- If the command result is false, then **UPS** runs the code following the **`Else`** statement and preceding the **`EndIf`** statement. If there is no **`Else`** statement, **UPS** does nothing.

Also see sections 35.3.4 *EndIf* and 35.3.3 *Else*.

A conditional **`If()...Else()...Endif()`** structure has no effect on dependencies. It may not work as you expect if you put **`If()...EndIf()`** around **`SetupOptional()`** and/or **`SetupRequired()`** statements. Developers writing table files that use **`If()`** statements to conditionally run setup commands must test thoroughly using **`setup -V`**, and must read the generated script files.

Syntax

`If(<command>)`

Example

Here is a standard example:

```
Action=Setup
ProdDir()
SetupEnv()
EnvSetIfNotSet(FOO,":")
EnvPrepend(FOO, ${UPS_PROD_DIR}/basic)

If( test -d ${UPS_PROD_DIR}/exciting )
  EnvPrepend(FOO, ${UPS_PROD_DIR}/exciting)
Else()
  EnvPrepend(FOO, ${UPS_PROD_DIR}/boring)
EndIf( test -d ${UPS_PROD_DIR}/exciting )
```

Here is an example showing behavior with dependencies. If **foo** depends on **bar** (i.e., **bar** appears in **foo**'s dependency list), then the code:

```
SetupRequired(foo)
If( some command that's true )
```

```
   SetupRequired(bar)
Else()
Endif( some command )
```

puts nothing inside the If statement in the generated script files, since the **`SetupRequired(bar)`** is redundant.

Similarly, if **foo** depends on **bar** v2, then:

```
If( some command )
   SetupRequired(bar v1)
Else()
   SetupRequired(bar v2)
Endif( some command )
```

sets up either **bar** v1 or *nothing*, since the second one is redundant. If you want different dependencies, you must use different stanzas in the table file.

35.3.18 pathAppend

Description

Append **`<value>`** to existing $PATH-like environment variable. Reversible (runs **`pathRemove`**).

It is better to append than prepend if you just want to provide a value in case one is not there. If you want to override any existing value, you should prepend.

Syntax

`pathAppend(<VARIABLE>, <value> [, <delimiter>])`

Example

`pathAppend(PATH, ${UPS_PROD_DIR}/bin)`

Appends the value `${UPS_PROD_DIR}/bin` to the $PATH variable using the default delimiter.

35.3.19 pathPrepend

Description

Prepend **`<value>`** to existing $PATH-like environment variable. Reversible (runs **`pathRemove`**).

It is better to prepend than append if you want to override any existing value. If you just want to provide a value in case one is not there, you should append.

Syntax

```
pathPrepend(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
pathPrepend(PATH, ${UPS_PROD_DIR}/bin)
```

Prepends the value `${UPS_PROD_DIR}/bin` to the $PATH variable using the default delimiter.

35.3.20 pathRemove

Description

Remove the string **<value>** from existing $PATH-like environment variable. Reversible (runs **pathAppend**).

Syntax

```
pathRemove(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
pathRemove(PATH, ${UPS_PROD_DIR}/bin)
```

Removes the value `${UPS_PROD_DIR}/bin` from the $PATH variable.

35.3.21 pathSet

Description

Set a $PATH-like environment variable (in **csh** family, setting a $PATH is different than setting other environment variables). No choice of delimiter offered. Reversible (runs **envUnset**).

If this gets set wrong, your $PATH could get deleted. (To recover from this problem, should it occur, simply run **setup setpath**.)

Syntax

```
pathSet(<VARIABLE>, <value>)
```

Example

```
pathSet(PATH, /afs/fnal.gov/ups/<prod1/v1_0/SunOS+5/bin: ...)
```

Sets the $PATH to the value given (sample value truncated after first delimiter for brevity).

35.3.22 prodDir

Description

Set the $<PRODUCT>_DIR environment variable to the root directory of the product instance. Reversible (runs **unProdDir**).

Syntax

```
prodDir()
```

35.3.23 setupEnv

Description

Set the $SETUP_<PRODUCT> environment variable so that product can later be unsetup. Reversible (runs **`unsetupEnv`**).

Syntax

```
setupEnv()
```

35.3.24 setupOptional

Description

Setup another **UPS** product as a dependency, do not fail if the product doesn't exist. Reversible (runs **`unsetupOptional`**).

Syntax

The syntax is similar to the command **`setup`**:

```
setupOptional("[<options>] <product> [<version>]")
```

Example

```
setupOptional("perl")
```

Setup the default instance of the product **perl**, if available. Do not fail if not found.

35.3.25 setupRequired

Description

Setup another **UPS** product as a dependency; fail if product not found. Reversible (runs **`unsetupRequired`**).

Syntax

The syntax is similar to the command **`setup`**:

```
setupRequired("[<options>] <product> [<version>]")
```

Example

```
setupRequired("-j Info")
```

Setup the default instance of the product **Info** and no dependencies; fail if not available.

35.3.26 sourceCompileOpt

Description

If **`<fileName>`** exists, then source it and skip remaining functions; otherwise just complete the remaining functions. This is typically used in conjunction with **`writeCompileScript`**; see section 35.3.38 *writeCompileScript*.

Syntax

```
sourceCompileOpt("<fileName>")
```

Example

```
sourceCompileOpt("/my/compile/script")
```

This sources the specified script which was created with **`writeCompileScript`**. If script doesn't exist, process continues.

35.3.27 sourceCompileReq

Description

Source **`<fileName>`** and skip all remaining functions; fail if file not found. This is typically used in conjunction with **`writeCompileScript`**; see section 35.3.38 *writeCompileScript*.

Syntax

```
sourceCompileReq("<fileName>")
```

Example

```
sourceCompileReq("/my/compile/script")
```

This sources the specified script which was created with **`writeCompileScript`**. If script doesn't exist, process fails.

35.3.28 sourceOptCheck

Description

Check if specified script exists and if so, source it and check return status for error. If error, abort script and return. Reversible (runs **`sourceOptCheck`** on the "un" script, e.g., `current` and `uncurrent`).

The functions **`execute`**, **`sourceOptCheck`**, **`sourceOptional`**, **`sourceReqCheck`**, and **`sourceRequired`** each take a required parameter (**`UPS_ENV_FLAG`**) which indicates whether to define **UPS** local variables. This parameter can take the following values:

UPS_ENV	define all local **UPS** environment variables before sourcing (the script or command relies on these being defined)
NO_UPS_ENV	do not define the local **UPS** environment variables (the script or command doesn't use them)

The functions **`sourceOptCheck`**, **`sourceOptional`**, **`sourceReqCheck`**, and **`sourceRequired`** each take an optional parameter (**`EXIT_FLAG`**). This parameter can take the following values:

CONTINUE	after sourcing the script, continue with the next function (the default)
EXIT	after sourcing the script, skip the rest of the functions

Syntax

```
sourceOptCheck(<SCRIPT>.${UPS_SHELL}, UPS_ENV_FLAG [,
  EXIT_FLAG])
```

Example

```
sourceOptCheck(${UPS_UPS_DIR}/current.${UPS_SHELL}, UPS_ENV)
```

Check if **`${UPS_UPS_DIR}/current`** exists. If so, first define all local **UPS** environment variables, then source the script and check return status for error. If error, abort script and return.

35.3.29 sourceOptional

Description

Check if **<SCRIPT>** exists and if so, source it. If script not found, continue. Reversible (runs **sourceOptional** on the "un" script, e.g., `current` and `uncurrent`).

See section 35.3.28 *sourceOptCheck* for information about the parameters **UPS_ENV_FLAG** and **EXIT_FLAG**.

Syntax

```
sourceOptional(<SCRIPT>.${UPS_SHELL}, UPS_ENV_FLAG [,
  EXIT_FLAG])
```

Example

```
sourceOptional(${UPS_UPS_DIR}/current.${UPS_SHELL}, UPS_ENV)
```

Check if **${UPS_UPS_DIR}/current** exists. If so, first define all local **UPS** environment variables, then source the script. If not, continue.

35.3.30 sourceReqCheck

Description

Source **<SCRIPT>** and check return status for error; fail if script not found. If error, abort script and return. Reversible (runs **sourceOptCheck** on the "un" script, e.g., `current` and `uncurrent`).

See section 35.3.28 *sourceOptCheck* for information about the parameters **UPS_ENV_FLAG** and **EXIT_FLAG**.

Syntax

```
sourceReqCheck(<SCRIPT>.${UPS_SHELL}, UPS_ENV_FLAG [,
  EXIT_FLAG])
```

Example

```
sourceReqCheck(${UPS_UPS_DIR}/current.${UPS_SHELL}, UPS_ENV)
```

Check if **${UPS_UPS_DIR}/current** exists. If not, it will fail. If script exists, first define all local **UPS** environment variables, then source the script and check return status for error. If error, abort script and return.

35.3.31 sourceRequired

Description

Source **<SCRIPT>**; fail if script not found. Return status not checked. Reversible (runs **sourceOptional** on the "un" script, e.g., current and uncurrent).

See section 35.3.28 *sourceOptCheck* for information about the parameters **UPS_ENV_FLAG** and **EXIT_FLAG**.

Syntax

```
sourceRequired(<SCRIPT>.${UPS_SHELL}, UPS_ENV_FLAG [,
  EXIT_FLAG])
```

Example

```
sourceRequired(${UPS_UPS_DIR}/current.${UPS_SHELL}, UPS_ENV)
```

Check if **${UPS_UPS_DIR}/current** exists. If not, it will fail. If script exists, first define all local **UPS** environment variables, then source the script.

35.3.32 unAlias

Description

Remove alias/function of specified name.

Syntax

```
unAlias(<NAME>)
```

35.3.33 Unless

Description

A conditional; to be used with **EndUnless** and optionally with **Else**, in the order
Unless(<command>)...Else()...EndUnless(<command>).
The **Unless** statement must include a command. If the command result is false, **UPS** executes statements that follow **Unless** and that precede either **EndUnless** or **Else**, whichever it encounters. If the command result is

true and an **`Else`** statement exists, **UPS** executes statements that follow **`Else`** and precede **`EndUnless`**. If the command is true and no **`Else`** statement exists, **UPS** does nothing.

See also sections 35.3.3 *Else* and 35.3.5 *EndUnless*. See section 35.3.17 *If* for information on dependencies; **`EndUnless`** works in an analogous manner.

Syntax

`Unless(<command>)`

Example

```
Action=Setup
ProdDir()
SetupEnv()
EnvSetIfNotSet(FOO,":")
EnvPrepend(FOO, ${UPS_PROD_DIR}/basic)

Unless( test -d ${UPS_PROD_DIR}/exciting )
  EnvPrepend(FOO, ${UPS_PROD_DIR}/boring)
Else()
  EnvPrepend(FOO, ${UPS_PROD_DIR}/exciting)
EndUnless( test -d ${UPS_PROD_DIR}/exciting )
```

35.3.34 unProdDir

Description

Unsets the $<PRODUCT>_DIR environment variable. Reversible (runs **`prodDir`**).

Syntax

`unProdDir()`

35.3.35 unsetupEnv

Description

Unsets the $SETUP_<PRODUCT> environment variable. Reversible (runs **`setupEnv`**).

Syntax

```
unsetupEnv()
```

35.3.36 unsetupOptional

Description

Runs **`unsetup`** on a product, does not fail if the product doesn't exist or if it's already unsetup. Reversible (runs **`setupOptional`**).

Syntax

The syntax is similar to the command **`unsetup`**:

```
unsetupOptional("[<options>] <product> [<version>]")
```

For previously setup products, the only options that are recognized include **`-e`**, **`-j`**, and **`-v`**.

Example

```
unsetupOptional("perl")
```

Unsets the default instance of the product **perl**, if already setup. Does not fail if product doesn't exist or has already been unsetup.

35.3.37 unsetupRequired

Description

Runs **`unsetup`** on a product; fails if product not found. Reversible (runs **`setupRequired`**).

Syntax

The syntax is similar to the command **`unsetup`**:

```
unsetupRequired("<options>] <product> [<version>]")
```

For previously setup products, the only options that are recognized include **`-e`**, **`-j`**, and **`-v`**.

Example

```
unsetupRequired("perl")
```

Unsets the default instance of the product **perl**, if already setup. Fails if product doesn't exist or has already been unsetup.

35.3.38 writeCompileScript

Description

Write a file of compiled functions for the given ACTION keyword value. It actually writes four files in total: `<script>.[c]sh` and `un<script>.[c]sh`.

The function **`writeCompileScript`** takes an optional parameter which can be one of the following:

OLD — if `fileName` exists, move the old one to `fileName.old` before creating the new one.

DATE — if `fileName` exists, move the old one to `fileName.{datestamp}` before creating the new one.

Syntax

```
writeCompileScript("<fileName>", "<ACTION>" [, OLD|DATE])
```

Example

```
writeCompileScript("/my/compile/script", "SETUP", OLD)
```

This executes the SETUP action and writes the output of the functions to the specified script, first saving the pre-existing script to `/my/compile/script.old`. This function knows to ignore the function **`sourceCompileReq`** or **`sourceCompileOpt`** if it encounters either at the top of the list of SETUP functions. See sections 35.3.26 *sourceCompileOpt* and 35.3.27 *sourceCompileReq*.

35.4 Functions under Consideration for Future Implementation

`copyCatMan`	Will copy catman files from source directory specified in table file by CATMAN_SOURCE_DIR to target directory specified in the **UPS** database configuration file by CATMAN_TARGET_DIR. Reversible (will run **`uncopyCatMan`**)
`copyHtml`	Will copy html files from source directory specified in table file by HTML_SOURCE_DIR to target directory specified in the **UPS** database configuration file by HTML_TARGET_DIR.
`copyInfo`	Will copy Info files from source directory specified in table file by INFO_SOURCE_DIR to target directory specified in the **UPS** database configuration file by INFO_TARGET_DIR.
`copyMan`	Will copy man files from source directory specified in table file by MAN_SOURCE_DIR to target directory specified in the **UPS** database configuration file by MAN_TARGET_DIR. Reversible (will run **`uncopyMan`**)
`copyNews`	Will copy news files from source directory specified in table file by NEWS_SOURCE_DIR to target directory specified in the **UPS** database configuration file by NEWS_TARGET_DIR.
`else ()`	Will begin an alternative branch
`elseif (<condition>)`	Will proceed to another condition
`endif ()`	Will end a conditional branch
`if (<condition>)`	Will begin a conditional branch

`uncopyCatMan`	Will remove catman files from target directory specified in the **UPS** database configuration file by CATMAN_TARGET_DIR. Reversible (will run **`copyCatMan`**)
`uncopyMan`	Will remove man files from target directory specified in the **UPS** database configuration file by MAN_TARGET_DIR. Reversible (will run **`copyMan`**)

35.5 Examples of Functions within Actions

35.5.1 A setup Action

This first example shows a **`setup`** action:

```
ACTION=SETUP
 prodDir()
 setupEnv()
 pathAppend(PATH, ${UPS_PROD_DIR}/bin)
 setupRequired("crow")
 setupOptional("gypsy")
```

When the product instance gets setup, **UPS** does five things in addition to **`setup`**'s internal processes:

- sets the variable $<PRODUCT>_DIR to the product root directory
- sets the variable $SETUP_<PRODUCT> to identify the product instance for unsetup
- appends the product's `bin` directory to the path
- sets up the product **crow** (and aborts the setup if a suitable current instance of **crow** is not available)
- sets up the product **gypsy**, if found (**`setup`** proceeds whether or not a suitable current instance of **gypsy** is available).

35.5.2 A "declare as current" Action

A second example illustrates steps for **UPS** to complete when the product instance is declared as current to the database:

```
ACTION=CURRENT
    execute("echo    Call    final    install    script    for
${UPS_PROD_NAME} ${UPS_PROD_VERSION}")
 sourceRequired(${UPS_UPS_DIR}/current, UPS_ENV)
```

UPS echoes the given text and sources the `current` script for the product.

35.6 Local Read-Only Variables Available to Functions

The read-only variables listed below are set by **UPS** and available for use with the functions described in section 35.3 *Function Descriptions*. In several functions, the flag UPS_ENV_FLAG controls whether these variables get set (see section 35.3.28 *sourceOptCheck*).

These **UPS** variables do not get exported to the environment, but exist only for the duration of, and in the context of, the processing of an action (actions are described in Chapter 34: *Actions and ACTION Keyword Values*). By contrast, the environment variables $<PRODUCT>_DIR and $SETUP_<PRODUCT> (described in section 23.1 *setup* under *Environment Variables Set by Default During setup*), if defined, remain set and available for use as long as the product is setup.[1]

35.6.1 List of Current Read-Only Variables

When you use these variables, always enclose them in curly brackets ({}) as shown in the list.

Local Read-Only Variable	Description of Value
${PRODUCTS}	Generally has the same value as the environment variable $PRODUCTS. The difference is that (read-only) ${PRODUCTS} keeps the value set at the time **UPS** was invoked, whereas (environment) $PRODUCTS may be reset. You can reset $PRODUCTS (i.e., using the function **`envSet (PRODUCTS, "<value>"`** in the table file) in order to use a new value in the temp file; $PRODUCTS won't get overwritten by ${PRODUCTS} as the temp file executes. See the example that follows this table. Note that this is not valid for the other read-only variables in this table; if you try to reset them (as environment variables), your values will get overwritten by the read-only values as the temp file executes.
${REQ_PROD_QUALIFIERS}	Requested product qualifiers (including optional ones), as opposed to the declared product qualifiers. For example, if you run **`setup fred -q opt1:opt2`** and fred is declared with QUALIFIERS="opt1", then ${UPS_PROD_QUALIFIERS} is "opt1", while ${REQ_PROD_QUALIFIERS} is "opt1:opt2".

1. The **`setup`** command and these variables are described in section 23.1 *setup*.

Local Read-Only Variable	Description of Value
${UPS_COMPILE}	Location and file name of a file containing compiled functions (see Chapter 38: *Use of Compile Scripts in Table Files*). It has the value of the combined keywords: COMPILE_FILE_DIR/COMPILE_FILE
${UPS_DIR}	Entire path to the `ups` directory of a product. (This is not the same as the environment variable $UPS_DIR that points to the root directory of the **UPS** product!)
${UPS_EXTENDED}	This set to 1 if the **`-e`** (extended) option was specified in the **`setup`** command (see section 25.2.1 *-e*)
${UPS_OPTIONS}	Option string that was passed with the **`-O`** (upper case **`O`**) flag (see Chapter 25: *Generic Command Option Descriptions*)
${UPS_ORIGIN}	This specifies the location of the master source files.
${UPS_OS_FLAVOR}	Operating system flavor as obtained from **`ups flavor`**
${UPS_OVERRIDE}	This contains the **UPS** command line option **`-H <flavor>`** that would override the default; not set by default. Can be used to "lie" to **UPS** about the flavor of the machine. May be expanded in the future to contain other **UPS** command line options.
${UPS_PROD_DIR}	Product instance root directory; same value as the environment variable $<PRODUCT>_DIR
${UPS_PROD_FLAVOR}	Product flavor chosen during instance matching
${UPS_PROD_NAME}	Product name as declared in the **UPS** database
${UPS_PROD_QUALIFIERS}	Product qualifiers chosen during instance matching. These are the qualifiers *declared with the selected instance*. They are not necessarily the same set of qualifiers specified on the command line via the **`-q`** option (the **UPS** matching algorithm chooses the “best fit” based on the specified qualifiers; not necessarily an exact match).
${UPS_PROD_VERSION}	Product version as declared in the **UPS** database
${UPS_SHELL}	Value can be csh or sh.
${UPS_SOURCE}	Value can be “source” for csh or “.” for sh
${UPS_THIS_DB}	Database in which this product instance is declared.
${UPS_UPS_DIR}	Path to the product instance’s `ups` directory
${UPS_VERBOSE}	This is set to 1 if the **`-v`** (verbose) option was specified (see Chapter 25: *Generic Command Option Descriptions*).

$PRODUCTS vs. ${PRODUCTS}: Resetting $PRODUCTS

This example is intended to illustrate the interaction between the read-only variable ${PRODUCTS} and the environment variable $PRODUCTS. There are a couple of potentially confusing points.

Let ${PRODUCTS} be set to `/fnal/ups/db`. Say in your table file you set $PRODUCTS to `/path/to/mydb` in the SETUP action, like this:

```
ACTION=SETUP
    envSet(PRODUCTS, "/path/to/mydb:${PRODUCTS}")
```

Now ${PRODUCTS} and $PRODUCTS are different. The following **`execute`** functions show the difference in the values. The function:

```
execute("echo $PRODUCTS", NO_UPS_ENV)
```

would produce:

`/path/to/mydb:/fnal/ups/db`

whereas the same function using ${PRODUCTS}, e.g.,

```
execute("echo ${PRODUCTS}", NO_UPS_ENV)
```

would produce only:

`/fnal/ups/db`

$PRODUCTS vs. ${PRODUCTS}: Effects on setup and ups depend

Another issue is the **`setup...`** functions. Say you have a product **fred v1_0** declared in `/path/to/mydb` (the database not included in ${PRODUCTS}). If you include a **`setupRequired`** or **`setupOptional`** function later in the SETUP action, e.g.,:

```
ACTION=SETUP
    envSet(PRODUCTS, "/path/to/mydb:${PRODUCTS}")
    setupRequired(fred v1_0)
```

the setup will fail because these functions only reference the read-only variable ${PRODUCTS}, which in this case doesn't include your product. You can get around this by using the **`execute`** function to set the product up:

```
execute ("setup fred v1_0", NO_UPS_ENV)
```

This function uses the environment variable $PRODUCTS.

Remember though, when you run a **`ups depend`** on a product, only products identified in **`setupRequired`** or **`setupOptional`** functions get listed. You would not see **fred v1_0** listed in the **`ups depend`** output for the main product in our example.

35.6.2 Read-Only Variables under Consideration for the Future

We plan to make the keyword values, listed in section 28.4 *List of Supported Keywords*, available as read-only variables available to functions. The read-only variable corresponding to a keyword will typically include “UPS_” prepended to it. E.g., the read-only variable corresponding to the keyword DECLARED will be ${UPS_DECLARED}. Several of these are already implemented in this way, e.g., ${UPS_PROD_DIR} corresponds to the keyword PROD_DIR.